GAO

United States General Accounting Office

Report to the Chairman, Subcommittee on Defense, Committee on Appropriations, House of Representatives

January 1993

SOFTWARE REUSE

Major Issues Need to Be Resolved Before Benefits Can Be Achieved

GAO

United States
General Accounting Office
Washington, D.C. 20548

Information Management and
Technology Division

B-251542

January 28, 1993

The Honorable John P. Murtha
Chairman, Subcommittee on Defense
Committee on Appropriations
House of Representatives

Dear Mr. Chairman:

The Department of Defense (DOD) estimates that expenditures for developing and maintaining software for its weapons, command and control, and other automated information systems currently exceed $24 billion a year. In an attempt to better manage these costs and improve its ability to develop and maintain high-quality software, Defense has initiated a comprehensive effort to incorporate software reuse practices into its software development efforts. Software reuse—the practice of developing new applications from existing software—offers the potential to greatly reduce the time, cost, and effort needed to develop and maintain high-quality software.

As requested by your office, this report provides background information on software reuse, including an overview of issues that can inhibit effective software reuse and information on Defense's strategy to implement a departmentwide software reuse program. Appendix I further details our objectives, scope, and methodology. Appendix II provides information on Defense's initiatives to incorporate software reuse into its software development process.

Results in Brief

Developing and maintaining software in organizations such as the Department of Defense is very costly. According to many experts in the software community, software reuse is a possible solution to reduce these costs, as well as to increase software productivity and reliability. Although these benefits and savings are compelling, achieving them will require the resolution of significant technical, organizational, and legal issues.

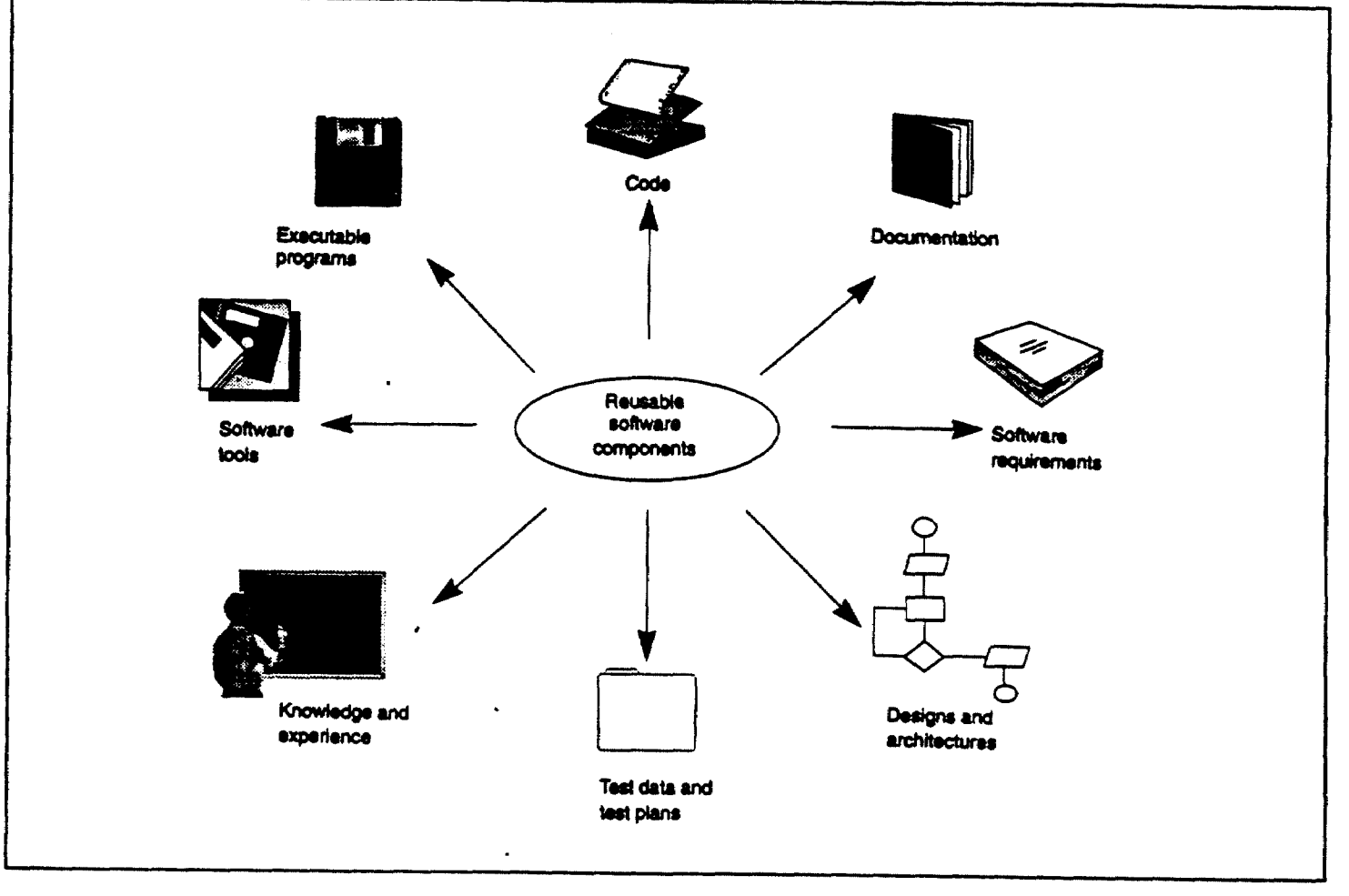
Even while proclaiming the potential of reuse, many software experts have questioned the maturity of software reuse. These experts indicate that methodologies to implement reuse have not been fully developed, tools to support a reuse process are lacking, and standards to guide critical software reuse activities have not been established.

Beyond such technical difficulties, organizations also face numerous challenges to effectively implement and practice software reuse. An organization must make a significant commitment to reuse because fundamental changes in the organization's software development approach will be needed and significant up-front costs for training and tools will be required. Further, uncertainties in legal policies, such as liability and intellectual property rights that currently hinder software reuse, need to be addressed, and acquisition policies need to be modified to better promote reuse.

Background

Software reuse is the practice of using existing software components to develop new applications. Reusable software components can be executable programs, code segments, documentation, requirements, design and architectures, test data and test plans, or software tools. They may also be knowledge and information needed to understand, develop, use, or maintain the component. Figure 1 shows examples of the different types of reusable software components.

Figure 1: Examples of Reusable Software Components



There are two basic forms of software reuse—opportunistic and systematic. Opportunistic reuse is practiced in an ad-hoc fashion during software development. In opportunistic reuse, new applications are developed from software that has been salvaged from existing systems and modified to meet the specific needs of that application. Systematic reuse is planned and integrated into a well-defined software development process. In systematic reuse, new applications are developed from software that has been designed and developed to be reused specifically for other similar applications.[1]

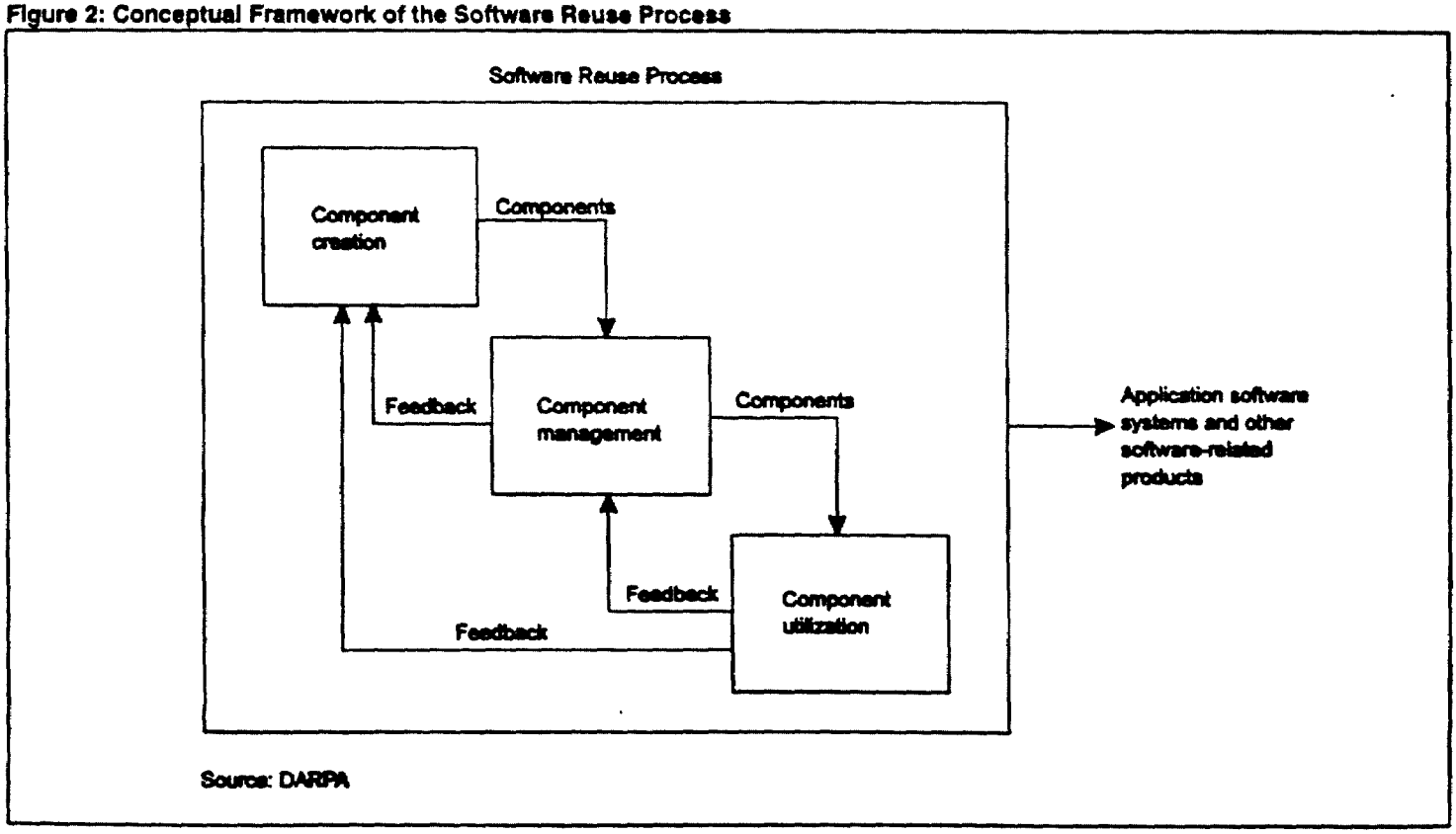
[1]In systematic reuse, new reusable software is also created as a by-product of applications development.

Software reuse can be practiced vertically or horizontally. Vertical reuse is the reuse of software components within a single domain.[2] For example, a software component that implements procedures to withdraw federal taxes from a paycheck can be reused by different accounting systems within the payroll application domain. Horizontal reuse, on the other hand, is the reuse of software components across different domains. For example, software components, such as sort and merge procedures, can be reused by systems in many application domains.

Software Reuse Process

The software reuse process consists of three stages: component creation, component management, and component utilization, as shown in figure 2.

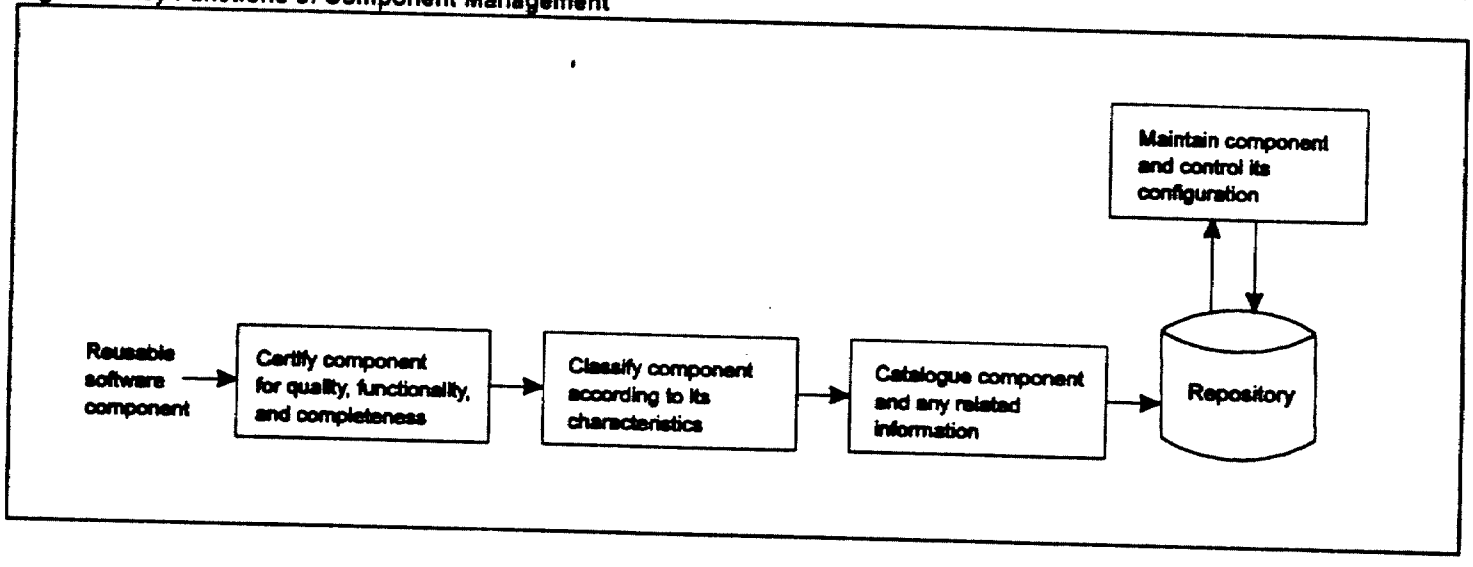
Figure 2: Conceptual Framework of the Software Reuse Process



[2]A domain is a family of related systems that exhibit common objects and operations.

This framework, established by the Defense Advanced Research Projects Agency's (DARPA) Software Technology for Adaptable Reliable Systems (STARS) program, presents the flow of information within the software reuse process and its products. During component creation, domains where reuse is possible are identified and reusable software components are developed. Once components are developed, they are stored and managed in a software repository, which is a library that allows users to access, search, and retrieve the components. Key functions of component management include certifying, classifying, and cataloguing components, as well as configuration control of the software components as a result of software upgrades and maintenance. Figure 3 illustrates the basic functions of component management.

Figure 3: Key Functions of Component Management



In component utilization, components in the repository are searched, evaluated, selected, and integrated into the software product under development. Components can be used to either develop application software systems or create new reusable components and software-related products.

Potential Benefits of Software Reuse

Systematic reuse is viewed as a possible means to reduce software development costs while improving software quality. According to a number of software experts, reuse has the potential to

- increase productivity by reducing the time and effort needed to develop software,
- increase reliability because systems will be developed with thoroughly tested and proven components,
- reduce costs by sharing knowledge and practices needed to develop and maintain software, and
- establish a more standard and consistent approach to software development and maintenance by using common components and procedures.

As an example, the Software Engineering Laboratory (SEL) at the National Aeronautics and Space Administration's (NASA) Goddard Space Flight Center achieved significant benefits by implementing software reuse in the development of software products in its Flight Dynamics Division. In a 1991 study, SEL reported

- a 35-percent reduction in effort needed to deliver a line of code (from .65 to .42 staff hours),
- a 53-percent increase in daily productivity (from 12.4 to 19 lines of code per day), and
- an 87-percent increase in quality (from 3.9 errors to .5 errors per thousand lines of delivered code).[3]

While the results of the SEL study appear promising, experts caution that the benefits of software reuse are not easily or quickly realized. The potential impact of software reuse remains questionable because of technical, organizational, and legal issues that need to be addressed.

Technical Issues

Establishing a systematic software reuse program is difficult. Few organizations—in either the private or public sectors—have been able to incorporate software reuse into their software development practices because the technical knowledge to develop and apply software reuse methodologies, standards, and tools is still evolving. Table 1 summarizes the technical barriers to software reuse discussed in the following sections.

[3]Proceedings of the Sixteenth Annual NASA/Goddard Software Engineering Workshop: Experiments in Software Engineering Technology, Software Engineering Laboratory, December 1991.

Table 1: Summary of Technical Issues

Domain analysis	-lack of standard methods to process information on domains -lack of standard methods to represent the outputs of domain analysis
Classification of software components	-no accepted standards to classify components -classification depends upon a domain analysis
Interoperability of software repositories	-lack of standards for interoperation of repositories
Adaptation of software components	-adaptation depends upon the availability of information on component -more required adaptation can offset the savings and benefits of software reuse
Reuse of systems designs and architectures	-designs and architectures are harder to represent because they are more abstract -lack of standards to represent designs and architectures -lack of tools to represent, develop, and maintain designs and architectures
Software metrics	-lack of standard metrics -inconsistent interpretation of metrics -collecting metrics is expensive and time-consuming

Domain Analysis

Domain analysis involves systematically gathering and representing information on software applications. Experts in the software community generally agree that domain analysis is at the "heart of reuse." Its purpose is to generalize common features in similar application areas, identify the common objects and operations in these areas, and define and describe their relationships. Once collected, the information can be used to create reusable software components that support these areas. For example, in an airline reservation system domain, common objects are flights and seats, while common operations include flight scheduling and seat assignments. These objects and operations are related in specific ways to the airline reservation system domain. As such, software components that support these objects and operations could be reused by developers of other airline reservation systems.

Domain analysis is a complex process that involves acquiring and representing knowledge on specific domains. Information on the domain must be identified, compiled, analyzed, and represented in a format so that it can be reused. The domain analyst needs to not only identify the objects and operations and their relationships in the domain, but also be able to explicitly represent that information so others can easily understand and reuse it.

However, standard methods to process and represent information on a domain are lacking. Current domain analysis methodologies, such as the Software Engineering Institute's Feature Oriented Domain Analysis (FODA) and Dr. Ruben Prieto-Diaz's Top-Down, Bottom-up Approach, are still evolving and thus do not completely address these functions.[4]

Classification of Software Components

Classification is a process of systematically grouping reusable software components stored in a software repository. A classification scheme for a software repository is analogous to the Dewey Decimal System for a library. Its purpose is to provide the basic organization of a repository so users can easily access, search, and retrieve components in the repository.

Establishing a classification scheme is knowledge-intensive and time-consuming. It requires combining the knowledge inherent in the components of the repository with the knowledge about the application domain where the components are going to be used. Common characteristics of the components are then grouped and organized into a structure that can be easily understood by repository users. While automated tools exist to catalogue software components (store and retrieve components in a repository), the key difficulty in classification is how to organize the overall repository because there are no accepted standards for classifying components.

Interoperability of Software Repositories

Interoperability is the ability of two or more systems to exchange information. It is an important capability in instances where multiple repositories exist because it permits software repositories to share components, reduce the number of redundant components in the different repositories, and make components available to all repository users.

Development is currently underway, for example, in DARPA's STARS program to establish an architectural framework for repository interoperability. However, standards for interoperability of software repositories, such as nomenclature, communication protocols, and component exchange formats, do not exist. Currently the Reuse Library Interoperability Group (RIG) is addressing standards for interoperability and plans to submit

[4]For more information on FODA, contact Dr. Sholom Cohen, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. For more information on the Diaz model, contact Dr. Ruben Prieto-Diaz, Reuse Inc., Fairfax, Va.

proposals to standards organizations, such as the Institute of Electrical and Electronics Engineering.[5]

Adaptation of Software Components

Adaptation involves modifying a software component to make it reusable in different software applications. It requires the software developer to determine what interfaces are needed and then tailor the component and/or application to make them operate together. Since the current state-of-practice is mainly opportunistic, most of the benefits that can be gained from software reuse are highly dependent on effective adaptation methods. However, adaptation is a difficult process because the developer has to understand

- how the component currently functions,
- how the new application works, and
- what modifications are needed to make the component work in the new application.

Without this information, a developer cannot easily adapt the software component for reuse. Even with the information, the adaptation process can be labor-intensive, potentially offsetting time and cost savings promised from software reuse.

Reuse of Systems Designs and Architectures

Although the current state-of-practice of software reuse has been mainly limited to the reuse of code, experts believe that the reuse of other software products, such as systems designs and architectures, can further increase the benefits of software reuse. They call this "higher-level reuse" because it involves reusing products that are from software development phases that occur prior to (or higher than) the one in which code is written.[6] According to these experts, the reuse of higher-level components will yield greater benefits because designs and architectures

- are more flexible than code because they are independent of language, hardware platforms, and implementation-specific details;
- represent application solutions rather than implementation solutions; and

[5]RIG is a volunteer organization composed of members from government, academia, and private industry. Membership is open to any organization interested in the interoperability of government-sponsored reuse libraries.

[6]In the traditional software development process, there are four successive phases: planning, design, coding and testing, and integration and testing. In the planning phase, requirements are set; during the design phase, designs and architectures are developed; in the coding and testing phase, code is written and tested; and in the integration and testing phase, the coded components are combined and tested as a whole.

- can be used to automatically create lower-level components, such as code.

However, formally representing systems designs and architectures in a reusable form is very difficult because they are not as tangible as code. Further, standards and tools to represent and develop systems designs and architectures are lacking.

Software Metrics

Software metrics are quantifiable measures that are used to assess the products and processes of software development. Such metrics may include measures of usefulness, cost, and quality that could be used to better manage software development programs. However, identifying and establishing metrics is difficult because standard methodologies do not exist to collect data for software development and products in general, or for reuse in particular. As such,

- current metrics are inconsistent,
- interpretation of metrics can vary from individual to individual, and
- collecting metrics is a very expensive and time-consuming process.

Without effective metrics, organizations cannot adequately determine the costs and benefits of incorporating software reuse into their software development processes.

Organizational Issues

Software reuse will not happen merely because the technical means for achieving it become available. Software experts told us that top management must be convinced to make a business decision to incorporate systematic reuse into the software development process. Further, project managers and software developers must be willing to make fundamental changes in the way they develop software. Otherwise, software reuse will remain at the opportunistic level, and the potentially greater benefits of systematic reuse will not be realized.

Gaining Management Support and Commitment

Experts have stated that for a software reuse program to be successful, top management must decide and commit to implementing a systematic reuse program throughout the organization. They noted that top management needs to

- incorporate software reuse practices into the software development process,

- train and educate employees on software reuse,
- develop and provide tools to practice software reuse, and
- allocate the proper funds and resources to support a reuse program.

However, experts generally agree that overall such management support is lacking. For example, at a jointly sponsored workshop by the Software Productivity Consortium, the Microelectronics and Computer Technology Corporation, the Software Engineering Institute, and the Rocky Mountain Institute on Software Engineering, attendees unanimously agreed that management generally has a short-sighted view on software development and is often not willing to commit resources to acquire needed tools and training in software reuse technology.

In addition, some experts believe that top management is hesitant to invest in software reuse because the benefits of software reuse are not quickly realized and are uncertain. To illustrate, some experts estimate that the savings from reusing a component will not be realized until that component has been reused at least three times and believe that it initially costs about 20 to 55 percent more to develop reusable software.

Gaining Support of Project Managers and Developers

Another common organizational issue is the unwillingness of project managers and software developers to develop and use reusable components. As noted above, developing reusable software is more costly and time-consuming. As such, project managers, who are often pushed by limited funds and tight schedules, have little incentive to allocate the additional time and resources needed to develop reusable software components.

Additionally, software developers are often reluctant to accept and use reusable components for fear that the components will not be as efficient, effective, or reliable as the software they write. Further, using reusable software components requires that the components be understood and adapted to meet the specific needs of a software system before it can be integrated. In either case, the reluctance of software developers to use reusable software and the lack of incentives for program managers to develop reusable software components remain issues that need to be addressed.

Legal Issues

The software community's hope for widespread reuse also brings about a number of challenging legal issues. The Institute for Defense Analysis (IDA)

recently published the proceedings of a 1990 workshop on legal issues, sponsored by the Strategic Defense Initiative Organization.[7] The workshop reached the following conclusions:

- Large-scale software reuse will likely cause more complex and more frequent encounters with legal issues.
- Large-scale reuse will require a national registry that tracks the source of original development and modifications for each component.
- A mechanism is needed to reward developers who modify and add value to existing components, while still protecting the rights of the original developer.
- It is not good policy and may conflict with federal law if the government assumes all legal liabilities associated with a reuse repository.
- Software patents and licensing policies need to be addressed.

Our discussions with reuse experts in the software community corroborated these concerns. Most believed that strategies are needed to address intellectual property rights, liability, and acquisition policies of reuse.

Intellectual Property Rights

Software is protected legally as intellectual property through laws that control its dissemination and use. These laws relate to the exclusive ownership of the idea, the form of expression of the idea, and the use of the idea and its expression. There are three basic methods to protect software: patents, copyrights, and trade secrets. Patents protect the rights to the idea itself, while copyrights protect the rights to the expression of the idea. Trade secret laws protect the rights to confidential business information.[8] However, in many cases laws are not clear about the enforcement of intellectual property rights. As such, a major challenge facing software reuse is to balance these rights between software suppliers, repositories, and users.

Several approaches have been proposed by various members of the software community to address intellectual property rights. For example, one approach proposed having repositories acquire full rights to software components. However, questions were raised about a repository's ability

[7]Proceedings of the Workshop on Legal Issues in Software Reuse (IDA Document D-1004), Institute for Defense Analysis, July 1991.

[8]Patents and copyrights are governed by federal law. Trade secrets are governed by state laws that may vary from state to state.

to motivate suppliers to do this because the component supplier will lose exclusive rights to commercially market the component.

Another approach proposed having software suppliers license limited software rights to a repository. However, issues were raised that if rights are transferred through a licensing agreement, future users of the component would need to be protected from breaching license agreements. Further, repositories would need to track all uses of software components to ensure that royalties and service fees are compensated to the component supplier and that licensing agreements are enforced. If the repository is unable to track software components and enforce licensing agreements, suppliers could be discouraged from giving up partial software rights to the repository.

Liability

Liability, in the context of software, refers to the legal responsibility for harm attributable to software components. Liability may affect not only the supplier of the component, but also the repository and user. For example, software suppliers could be liable for submitting defective software components that fail to meet performance standards or cause a software system malfunction. Repositories could be liable for marketing and distributing defective components or not properly enforcing the rights to software components. Users could be liable for infringing the intellectual property rights attached to a software component.

However, the subject of liability for software is fairly new to the law. As a consequence, there are still questions, such as whether software is a product or service, that have left some uncertainty about the nature of liability that may accompany software. For this reason, experts believe that organizations interested in reuse need to address liability issues, otherwise

- suppliers may be reluctant to submit components for reuse,
- repositories may limit the availability of components, and
- users may be unwilling to use the components in the repository.

Acquisition Policies

Many in the software reuse community acknowledge that changes need to be made in federal acquisition policies of software systems before software reuse can be effective. There are concerns that if industry is not involved in these efforts, reuse goals will not be achieved. Some of the issues that have arisen include how reuse should be considered in the

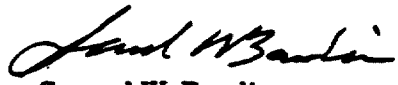
request for proposals process, what criteria to use to evaluate proposals, how costs of reuse will be evaluated and estimated, and what incentives are needed in solicitation documents to promote reuse. These concerns have prompted the actions of groups such as the Special Interest Group Ada and Institute for Defense Analysis, which sponsored a workshop to determine how and what to incorporate into the procurement process to encourage and promote reuse.

As requested, we did not provide a draft of this report to the Department of Defense. Instead, we discussed the facts of this report with officials from the Office of the Director for Defense Information; the Defense Information Systems Agency's Center for Information Management; the Air Force, Army, and Navy; and with software experts in industry. These officials generally agreed with the facts as presented. We have incorporated their views in the report as appropriate.

We conducted our review between April 1992 and December 1992, in accordance with generally accepted government auditing standards. As agreed with your office, unless you publicly announce the contents of this report earlier, we plan no further distribution until 30 days from the date of this letter. We will then send copies of this report to other interested committees; the Director for Defense Information; the Director for Defense Research and Engineering; and other interested parties. Copies will also be made available to others upon request.

If you or your staff have any questions concerning this report, please contact me at (202) 512-6240. Other major contributors are listed in appendix III.

Sincerely yours,



Samuel W. Bowlin
Director, Defense and Security
Information Systems

Contents

Abbreviations

ASDC3I	Assistant Secretary of Defense for Command, Control, Communications and Intelligence
CARDS	Central Archive for Reusable Defense Software
DARPA	Defense Advanced Research Projects Agency
DDR&E	Director for Defense Research and Engineering
DISA	Defense Information Systems Agency
DLA	Defense Logistics Agency
DOD	Department of Defense
GAO	General Accounting Office
FODA	Feature Oriented Domain Analysis
IDA	Institute for Defense Analysis
IMTEC	Information Management and Technology Division
NASA	National Aeronautics and Space Administration
RIG	Reuse Library Interoperability Group
SEL	Software Engineering Laboratory
STARS	Software Technology for Adaptable Reliable Systems

Appendix I

Objectives, Scope, and Methodology

On February 7, 1992, the Chairman of the Subcommittee on Defense, House Appropriations Committee, requested that we provide background information on software reuse, including an overview of issues that can inhibit effective software reuse, and information on Defense's strategy to implement a departmentwide software reuse program.

To meet our objectives, we

- met with reuse experts in private industry, government, and academia to discuss the concepts of reuse, including benefits and issues of effective reuse;
- attended the 5th Annual Software Reuse Workshop in Palo Alto, Ca., and reviewed papers to identify the state of reuse;
- observed and discussed reuse experiences with private industry, private organizations, and special interest groups;
- met with Defense officials to identify roles, responsibilities, and strategies for Defense's departmentwide software reuse initiative; and
- examined reuse activities and research and development efforts underway in Defense.

We performed our work at the Office of the Director for Defense Information, Arlington, Va.; Center for Information Management, Arlington, Va.; Defense Advanced Research and Projects Agency, Arlington, Va.; Software Engineering Institute, Pittsburgh, Pa.; U.S. Army Software Reuse Center, Washington, D.C.; U.S. Navy Information Systems Management Center, Washington, D.C.; U.S. Air Force Systems and Software Design, Hanscom Air Force Base, Ma.; Defense Logistics Agency's Systems Automation Center, Columbus, Oh.; and National Aeronautics and Space Administration's Goddard Space Flight Center, Greenbelt, Md.

We also visited the offices of International Business Machines Corporation in Gaithersburg, Md., Manassas, Va., Rockville, Md., and Owego, Ny; University of Maryland, College Park, Md.; Software Productivity Consortium, Herndon, Va.; Reuse, Inc. Fairfax, Va.; Raytheon Missile Systems Division, Bedford, Ma.; Mitsubishi Electric Research Laboratories, Cambridge, Ma.; Westinghouse Electric Corporation, Baltimore, Md.; Massachusetts Institute of Technology, Boston, Ma.; and Hewlett Packard, Palo Alto, Ca.

Department of Defense Reuse Initiative

Over the last few years, software reuse has gained increased attention throughout the Department of Defense as a way to reduce software costs and improve productivity and software quality. A draft of Defense's software technology strategy states that the greatest estimated Defense cost savings over the next 15 years will come from reusing software assets—a savings of $11.3 billion in constant 1992 dollars by the year 2008.[1] Other Defense documents note that the benefits of reuse go beyond cost savings to include substantial increases in productivity from avoidance of rework, and added software quality through the use of tested components.

Responsibility for software within the Department of Defense is divided between the Director for Defense Research and Engineering (DDR&E), who is responsible for embedded systems and information technology research, and the Assistant Secretary of Defense for Command, Control, Communications and Intelligence (ASDC3I), who carries responsibility for information systems and command and control systems.

The Defense Software Reuse Initiative

A Memorandum of Agreement between ASDC3I and DDR&E, effective November 25, 1991, established a cooperative partnership for implementing software and other information technology initiatives for the Department of Defense. On the basis of this agreement, the Director for Defense Information proposed a Defense software reuse initiative to provide a "single, consistent departmentwide software reuse strategy, with associated policies, practices, approaches, and programs." The initiative sought to build partnerships among users of reusable components, suppliers of these components, and the research and development community.

The Defense software reuse initiative is a voluntary and cooperative alliance of individual DOD reuse activities with active participation from the three major software reuse programs: Air Force's Central Archive for Reusable Defense Software (CARDS), DARPA's Software Technology for Adaptable Reliable Systems (STARS), and the Defense Information Systems Agency's (DISA) Software Reuse Program. It is guided by a software reuse executive steering committee representing the ASDC3I, DDR&E, Joint Staff, Army, Navy, Air Force, DISA, Defense Logistics Agency (DLA), Defense Intelligence Agency, and the National Security Agency. The steering committee reports to both ASDC3I and DDR&E, and is supported by working groups responsible for addressing technical and management issues. DISA's

[1]Draft of Department of Defense Software Technology Strategy, Director of Defense Research and Engineering, December 1991.

Center for Information Management is managing the initiative and providing a focal point for coordination.

Initiative's Vision and Strategy

Defense's software reuse initiative holds a vision, "to drive the DOD software community from its current 'reinvent the software cycle' to a process-driven, domain-specific, architecture-centric, library-based way of constructing software." The strategy for achieving this vision lies in systemizing the reuse process by identifying opportunities for reuse and establishing a process to capitalize on those opportunities. Defense details 10 elements of this strategy:

- Specify the domains where reuse opportunities exist and identify criteria to prioritize, qualify, and select domains for application of reuse techniques.
- Define the types of products suitable for reuse and develop criteria to validate these components for new applications.
- Determine what ownership criteria pertain to these components and require conscious decisions regarding their ownership.
- Modify the current acquisition process so reuse is integrated into each phase of the acquisition process and into the overall system/software life cycle.
- Define models that may suggest novel strategies and require tailored acquisition approaches to support reuse, in order to guide business decisions.
- Establish procedures to collect metrics that (1) measure the payoff from the reuse initiative and (2) aid developers in the selection of reusable components.
- Define standards for the various types of components that will permit their certification for reuse.
- Pursue a technology-based investment strategy that identifies, tracks, and transitions appropriate reuse-oriented process and product technologies.
- Conduct comprehensive training to ensure that practitioners and policymakers capitalize on the initiative.
- Exploit near-term products and services that facilitate movement to a reuse-based paradigm.

Major Contributors to This Report

Information Management and Technology Division, Washington, D.C.	Frank W. Deffer, Assistant Director David Chao, Technical Adviser Joseph J.H. Cho, Evaluator-in-Charge Colleen M. Phillips, Senior Evaluator Wiley E. Poindexter, Jr., Senior Evaluator